

UNITED STATES PATENT APPLICATION

INVENTORS:

JOHN M. QUERNEMOEN

APPLICATION:

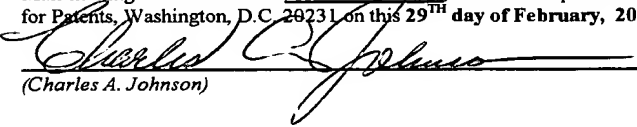
SIZING SERVERS FOR DATABASE MANAGEMENT  
SYSTEMS VIA USER DEFINED WORKLOADS

ATTORNEY DOCKET NO.

RA-5244

Charles A. Johnson  
Attorney for Applicant  
Reg. No. 20,852  
Telephone No. (651) 635-7702  
Unisys Corporation  
M.S. 4773  
PO Box 64942  
St. Paul, MN 55164-0942

CERTIFICATE UNDER 37 CFR 1.10: The undersigned hereby certifies that this transmittal letter and the paper of papers, as described hereinabove, are being deposited in the United States Postal Service, "Express Mail Post Office to Addressee" having an Express Mail mailing label number of **EK084997225US**, in an envelope addressed to: Box PATENT APPLICATION, Assistant Commissioner for Patents, Washington, D.C. 20231 on this 29<sup>TH</sup> day of February, 2000.

  
(Charles A. Johnson)

February 29, 2000  
(Date)

1c690 U.S. PTO  
09/515310  
02/29/00

**SIZING SERVERS FOR DATABASE MANAGEMENT SYSTEMS  
VIA USER DEFINED WORKLOADS**

Cross-Reference to Related Applications

*Inv. A1* → The present application is related to U.S. Patent Application Serial No. \_\_\_\_\_, filed \_\_\_\_\_, entitled DATABASE SIZER FOR NT SIZER SYSTEM; U.S. Patent Application Serial No. \_\_\_\_\_, filed \_\_\_\_\_, entitled COMBINATION OF MASS STORAGE SIZER, COMPARATOR, OLTP USER DEFINED WORKLOAD SIZER, AND DESIGN TRADE-OFF TOOL IN ONE PACKAGE; U.S. Patent Application Serial No. \_\_\_\_\_, filed \_\_\_\_\_, entitled BUILT IN HEADROOM FOR AN NT SYSTEM SIZER; U.S. Patent Application Serial No. \_\_\_\_\_, filed \_\_\_\_\_, entitled ALGORITHMS TO CALCULATE MASS STORAGE REQUIREMENTS FOR NT SIZER; and U.S. Patent Application Serial No. \_\_\_\_\_, filed \_\_\_\_\_, entitled METHOD OF COMPARISON FOR COMPUTER SYSTEMS AND APPARATUS THEREFOR, all of which are assigned to the assignee of the present invention and incorporated herein by reference.

Field of the Invention

The present invention is related generally to computers and data processing. More specifically, the present invention is related to methods for determining required sizes for servers based on user-supplied inputs.

Background of the Invention

Relational databases came into common use in computers over twenty years ago. Despite improvements in database software and new methodologies, relational databases remain the mainstay of database management systems (DBMS). Hardware vendors originally supported proprietary database management systems which ran primarily on machines manufactured by the hardware vendor. Software developers

later developed database management systems that were more open and ran on computers made by several vendors. The database management systems were also ported to run under various operating systems. This gave the advantage of spreading the cost of development over more sites and also uncoupled the dependence between  
5 hardware vendors and software vendors. Third party support and training also became more common.

Database management systems also became separated into client-side software and server-side software. This meant that the server-side software was decoupled from software having to do with the display, use, and formatting of the data received  
10 from the database. In particular, server-side software often handled mostly queries of existing data along with updates of existing data and insertion of new data.

Modern electronic commerce such as commerce over the Internet or business-to-business electronic commerce has placed increased demands on many servers. This has also made frequent upgrades necessary. Company mergers and acquisitions  
15 frequently make it necessary to incorporate large amounts of data from unexpected sources. Customer expectations also make it necessary to upgrade hardware to keep up with the faster response times users expect even though system loads may be increasing as well.

When upgrading or replacing database servers it is necessary to have a good  
20 idea as to the transaction handling requirements to be met. It is also desirable to know the DBMS transaction handling processing capabilities of any proposed system. It may be necessary to come up with a good estimate of the transaction handling capabilities of a proposed system. This may be particularly needed in a short time period, as during bid evaluations, during sales presentations, or repeatedly during  
25 scenario building.

In some situations, a required or desired transaction handling capability is based mainly on the capability of a known system. It may be the case that a given brand name DBMS server is believed to satisfy a current or future requirement, and the transaction processing capability of that DBMS server for a given benchmark is available from the vendor. A transaction rate could, in theory, be derived from a series of more specific user supplied information or requirements. It may be the case that the user has a more specific idea about what the requirements are for a system, such as detailed transaction specific information.

What would be desirable is a method to determine the required size for a database management system server based on given user supplied inputs. What would be advantageous are methods capable of determining required server size based both on required transaction processing benchmarks and on more detailed SQL processing requirements.

#### Summary of the Invention

The present invention includes methods for determining the required size for database management system servers. In one set of methods, the transaction rate requirement is input by a user as a transactions per minute requirement, where transactions per minute is the benchmark value specifying the number of New Order transactions per minute of the TPC-C benchmark, developed by the Transaction Processing Council (TPC), a consortium of hardware and software vendors. In one method, the server type or family of servers is input along with the maximum desired processor utilization and the transactions per second requirement. In some embodiments, a baseline system transactions per second is also input, to allow for comparisons with the later calculated values.

Algorithms, according to the present invention, operate on the previously

discussed inputs, and output the number of processors required, the effective CPU utilization and the number of users supported. The mass storage requirements and memory requirements can also be calculated. The transactions per second ratio of the input transactions per second to the baseline transactions per second can also be  
5 calculated along with the effective transactions per second.

Another set of methods according to the present invention allow the user to provide detailed inputs to allow the specification of a transaction rate based on more thorough knowledge of the make up of the transactions to be processed on the system. In one method, the transactions that are to make up the workload of the DBMS are  
10 named, the expected execution rate specified, and the transaction composition in terms of workload contribution is input.

The transaction composition workload can, in turn, be calculated by naming and specifying each type of SQL statement expected along with the number of records the SQL statement will operate on. In the case of SQL insert, delete, and update  
15 statements, the number of records is specified. In the case of SQL select statements, a selectivity criteria is specified. The selectivity criteria can vary depending on the type of select statement. If the select is a single table select, the selectivity input is the number of rows selected from the table. For selects involving the joining of two or more tables, the selectivity criteria was geared toward most applicable cases for  
20 online transaction processing. Adding more capability for selectivity in these cases does not go outside the scope of this invention. For the invention instance implemented here, the following selectivity criteria were used. If the select is a two-table join, then the selectivity criteria is the number of rows selected for in the outer, or left table, the inner table selectivity is assumed to be an average of four for each  
25 row selected from the outer table. If the select is a three-table join, then the selectivity

criteria is the number of values in the where clauses that pertain to the outer table. Joins of more than three tables were not included in this instance since these cases do not comprise the majority of transactions for this type of processing environment.

In the detailed input embodiment, the workload contribution of each SQL statement is calculated based on the SQL statement type and number of records and/or the selectivity criteria. The workload contributions of each SQL statement and the number of each type of such SQL statements can be summed to obtain a total workload contribution for the single transaction made up of these SQL statements. The workload contribution for each transaction can then be multiplied by the expected transaction execution rate for that type of transaction, and then a total workload contribution calculated for that transaction. This process can be repeated for each transaction to arrive at a total workload requirement for the transaction processing system. This total workload requirement can be expressed as a transaction throughput requirement, expressed in transactions per second, and used as input in the previously discussed method for determining the required size of the DBMS server. The calculated size of the DBMS server can thus be calculated with some precision.

#### Brief Description of the Drawings

Figure 1 is a highly diagrammatic schematic of a computer system including a database server;

Figure 2 is a highly diagrammatic view of a relational database;

Figure 3 is a highly diagrammatic view of a B-tree index for a relational database;

Figure 4 is a modified dataflow diagram of the method used to calculate DBMS server hardware requirements based on a workload requirement specified explicitly via user entered transactions per second; and

Figure 5 is a modified dataflow diagram of the method used to calculate a DBMS server hardware requirements based on a user defined workload specifying transactions, rate, and composition of user defined transactions and rates including SQL statements and types including the number of records operated on and selectivity criteria.

#### Detailed Description of the Invention

Figure 1 illustrates generally a database server system 20 including a server 22 supported by a CRT 24 and a printer 26 for programming, display, maintenance, and general Input/Output uses. Within server 22 is illustrated several CPU sockets 30 and 32, with CPU sockets 30 being populated with CPUs and CPU sockets 32 remaining empty for future expansion and population. Server 22 also includes a memory portion 40 which can contain a sufficient quantity of Random Access Memory (RAM) to meet the server's needs. A disk 28 is illustrated for mass storage, which can include disk drives or any other technology capable of holding the contents of the databases or databases to be managed. Several Network Interface Cards (NICs) 42 are illustrated as part of server 22 and are coupled to a network illustrated by a network link 36 which can be any communication link including Local Area Networks, Wide Area Networks, Ethernet, and the Internet.

Also connected to data link 36 are client computers 38. Software clients can, in fact, reside on the same machine as the server, but in common practice, the client processes usually run on a different machine. In one embodiment, server 22 is a computer running on the Microsoft NT operating system and clients 38 are smaller computers running a Microsoft Windows operating system.

Server 22 is preferably scaleable, having extra socketed capacity for memory, processors, NICs, and disk drives. This allows extra CPUs, memory, NICs, and mass

storage such as disk drives to be initially set to meet current needs and later expanded to meet changing needs.

Servers such as server 22 often exist to contain and manage data bases, such as those contained within relational database management systems (RDBMS). RDBMS  
5 include tables formed of rows or records and columns. Figure 2 illustrates an RDBMS table 50 formed of several columns 52 and several rows or records 54. Columns 52 typically include both fixed length or width columns and variable length or width columns, where the variable length may be allocated out of a common buffer elsewhere outside of the table itself. In practice, an RDBMS system has numerous  
10 tables to be stored and managed.

It is possible for rows 54 to be ordered according to one of the columns. In practice however, the records are typically not ordered, but are linked to indices that are ordered. In a simple example, one of columns 52, such as column 56, may contain a person's social security number and be linked via a software link 58 to an ordered  
15 index 60 which contains a sorted list of social security numbers along with the record number at which the corresponding record resides. Such a sorted list of only one column of table 50 can shorten a search from order  $n$  to order  $\log n$ . Such a sorted list still requires sorting upon the addition, deletion, and change of data.

A commonly used index method is illustrated for column 62, which is linked  
20 via a software link 64 to a B-tree index 66. B-tree 66 can contain a multi-level tree well known to those skilled in the software arts. B-tree 66 can be a multi-way tree such as an AVL tree or a 2-3 tree. B-tree indices have the advantage of being quick and easy to modify, without requiring massive amounts of sorting such as in a linear sorted index such as index 60. In particular, a B-tree can be maintained in a balanced  
25 condition with the addition of data to avoid skewing the shape of the tree.





values the tree is ordered for and points to nodes in the next level related to those keys or column values. For example, in a two-way tree, a node would have one index value, and two pointers, indicating which nodes to go to for values greater than or less than the index value, respectively. B-Trees and databases vary in what they have at the failure level. In some databases, herein termed “physical ordered databases”, the failure level has the records themselves linked together. In these databases, once the failure level is reached, the record has been obtained, with no further I/O necessary to obtain the record. In other databases, herein termed “non-physical ordered databases”, the nodes at the failure level contain only pointers or record numbers into the main table. In these databases, another I/O is required to obtain the record of interest. In some databases, the failure level contains neither the record of interest nor a pointer to the record of interest. Instead, a unique key is contained, requiring a search on that key to obtain the record of interest. For example, a search of B-Tree ordered on last name plus first name may return only a social security number upon successful completion. Another B-tree or highly optimized index based on social security number can then be rapidly searched for the record of interest. In this scheme, at least one more I/O is required after the failure level has been reached. The number I/Os required to reach a record is of interest to because it determines in part the speed of the database. Both disk I/O and network I/O require latent time to process.

In sizing a database, the RDBMS typically has a page size, or an aggregate unit of mass storage typically numbering thousands of bytes. The page size on some computers may be determined in part by operating system and disk system factors. The page size may also be determined by a desire to keep the width of internal variables within the database to manageable limits. The page size is fixed in some

RDBMSs and selectable in other RDBMSs.

The amount of mass storage required for a single table is a function of several variables, such as the number of rows or records and the number of columns. The database storage required is not a simple calculation of the row size and column sizes  
5 for several reasons. First, the column sizes or widths may be variable. Second, the page size enters into the calculation in a non-continuous manner, as some database allocation such as record allocation must lie within a single page rather than cross page boundaries, with some space wasted as a result. Third, some space in a page is set aside for future expansion or reserved for use a buffer space, as when re-ordering  
10 data. Fourth, not all space within a page is available for end user data storage, with some being used by the RDBMS itself or for other overhead. In particular, in some RDBMSs, a fraction of each page is specified as not available for initial storage. In some RDBMSs, a number of rows are set aside as non-usable. In some RDBMSs, a fraction of each record is set aside as non-usable. As previously mentioned, the size  
15 of the indices may be a large portion of table storage even though the data itself may not be stored within the indices. All of the aforementioned factors makes sizing the required databases a complicated matter, as is dealt with below.

Figure 4 illustrates a method 200 for determining the required size of a DBMS server using a transaction throughput benchmark as one of the inputs. In the  
20 embodiment illustrated, the TPC-C benchmark is used as the transaction processing environment. The tpmC input refers to the number of New Order Transactions Per Minute for the TPC-C benchmark. In TPC-C, throughput is defined as how many new order transactions per minute a system generates while the system is executing four other transactions types, payment, order-status, delivery, and stock-level; in this  
25 benchmark the New Order transaction comprise approximately 40% of the workload.

Method 200 includes inputs 202, outputs 204, and the effective transactions per second (TPS) 206. TPS 206 is the total number of transactions executed per second and is calculated using the equation:

$$\text{TPS} = \text{tpmC} / 36$$

5 Inputs 202 include the server type 208, the maximum desired processor utilization in percent 210, and the tpmC requirement 212. Optionally, the method may include the tpmC handling ability of a baseline system, indicated at 214, for purposes of comparison. Server type 208 may also be referred to a server family as the server type may include a configurable system, an expandable system, or a family  
10 of different hardware systems having varying hardware characteristics. In one embodiment, server type 208 may include varying numbers of processors (CPUs), memory, and mass storage.

The TPC-C benchmark is an Online Transaction Processing (OLTP) benchmark. Key characteristics are a high frequency of transactions executed, a large  
15 proportion of the transactions are inserts and updates, and transaction response times should be one second or less. Thus, these characteristics require indexed database access to minimize response time. Further, the database is expected to grow in proportion to the transaction rate.

A series of measurements were conducted with different numbers of processor  
20 configurations and processor speeds. Database sizes were proportionately increased with expected increase in transaction throughput. All benchmark tests were run to 100% processor utilization for the given configuration. The achieved throughput (tpmC), memory used, number of processors, processor speed, mass storage used, number of user simulated were recorded for each benchmark test.

25 Using the results of these tests a matrix of the above performance and

configuration values for each configuration is built. Where configurations are missing interpolation techniques are used to supply the missing values.

For example the following set of published measurement results were used:

<b>System</b>	<b>Mhz</b>	<b>nCPU</b>	<b>tpmC</b>	<b>Mass Storage</b>	<b>No. Users</b>	<b>Memory</b>
Unisys / Aquanta ES2025 Server	550	2	10266	705	8160	1024
Unisys / Aquanta ES2043 Server	500	4	23190	1593	18600	4096
Unisys / Aquanta ES2043R Server	500	4	23190	1593	18600	4096
Unisys / Aquanta ES2045 Server	500	4	23852	1636	19050	4096
Unisys / Aquanta ES5045 Server	500	4	24329	1815	19480	4096
Unisys / Aquanta ES2085R Server	550	8	37757	3079	30480	4096
Unisys / Aquanta ES5085R Server	550	8	40670	3562	32550	4096

- 5 This data was then averaged for each configuration and scaled to the lowest Mhz rating. This yielded the following table:

<b>Mhz</b>	<b>nCPU</b>	<b>tpmC</b>	<b>Mass Storage</b>	<b>No. Users</b>	<b>Memory</b>
500	2	9333	641	7418	1024
500	4	23640	1659	18933	4096
500	8	35649	3019	28650	4096

Using curve fitting, this data was then expanded to include estimates at 100 percent processor busy from one to the maximum possible processors. For 500 Mhz

- 10 processors, the results were as follows:

nCPU	tpmC	Mass Storage	No. Users	Memory
1	6,189	352	4,902	512
2	9,333	641	7,418	1,024
3	16,486	1,150	13,175	1,024
4	23,640	1,659	18,933	4,096
5	26,642	1,999	21,362	4,096
6	29,644	2,339	23,791	4,096
7	32,647	2,679	26,221	4,096
8	35,649	3,019	28,650	4,096

Using scaling techniques, similar results can be calculated for the 550 Mhz processors.

5 In the method 200 the smallest configuration satisfying both the tpmC 212 requirement and the MaxProcessorUtilization% 210 is selected. The configuration selected then yields the outputs #CPURequired 216 and MemoryRequirements,MB 220. The EffectiveCPUUtilization 218 is calculated as

EffectiveCPUUtilization 218 = (tpmCRequirement 212) / (tpmC at 100% for  
10 this configuration). The outputs MassStorageRequirement 220 and #UsersSupported 224 are obtained by interpolating between values for the configurations in the same proportion as tpmC required. For example, if 2 CPUs are required, then the #UsersSupported 224 is calculated as

#UsersSupported = (#UsersSupported with 1 CPU) +  
15 (EffectiveCPUUtilization 218) \* ((#UsersSupported with 2 CPUs) - (#UsersSupported with 1 CPU))

A similar calculation is made for MassStorageRequirement 222.

IAS. A2

00545310-022900

The mass storage required 222 can also be determined by method 200. The mass storage required can be satisfied by adding the appropriate size and number of mass storage devices such as disk drives. The required mass storage can be calculated using the methods described in related applications U.S. Patent Application Serial No. \_\_\_\_\_, filed \_\_\_\_\_, entitled ALGORITHM TO CALCULATE MASS STORAGE REQUIREMENTS FOR NT SIZER, and U.S. Patent Application Serial No. \_\_\_\_\_, filed \_\_\_\_\_, entitled DATABASE SIZER FOR NT SIZER SYSTEM, herein incorporated by reference. The tpmC ratio 226 provides a means for comparing the system specified by the inputs to method 200 and a baseline system tpmC 214 provided by the user. In one embodiment, a top window in the application program can be used to selectively browse through the tpmC database. The browsing can be filtered based on the operating system and database management system. In one embodiment, the tpmC database is filtered according to operating system and DBMS to provide a shorter list, which is selectable via a drop down list to a single tpmC database record, which can be termed the "baseline" system. The tpmC value from the record can be used as the value for the baseline system. In some embodiments, a known system from one vendor is selected as the baseline system on one part of the screen. The requirements can be used to select for another system, termed the 'target' system, from a second vendor in another part of the screen. The tpmC of the target system can be compared to the tpmC of the baseline system using ratio 226.

Referring now to Figure 5, in another embodiment of the invention, more detailed inputs are provided to the program to allow direct calculation of the estimated system configuration requirements. In Figure 5, a method 250 is used to calculate a workload contribution 258 for each user defined translation. WorkloadContribution

258 can be quantified in units of total CPU utilization and NIC utilization. A user defined transaction can include a transaction name, indicated at TxnName 260, an expected execution rate of the transaction, indicated at TxnRate 262, a LANSpeed, indicated at LANSpeed 298, and a transaction composition, indicated at TxnComposition 264. TxnName 260 can be a user-defined name given to each of the user-defined transactions. TxnRate 262 is the expected execution rate of the user-defined transaction, typically in transactions per second. TxnComposition 264 includes the user specified SQL composition of the transaction, discussed further below.

10 TxnComposition 264 can be determined by a method 252, and includes the SQL statement compositions 254, including the numbers of those SQL statements, and some parameters included for each SQL statement. SQL statements contributions include insert contributions 268, delete contributions 276, update contributions 278, and select contributions 284, each of which can include the seconds of total CPU time required to execute the SQL and the NIC usage in the total number of bytes passed both ways on the LAN. Method 252 can sum together the workload contributions of each of the SQL statements into total TxnComposition 264. An additional input to method 252 is server type 302 which provides the relative speed with which the SQL will be executed.

20 One method, indicated at 266, calculates the workload contribution of each SQL insert statement, which can be user defined to include a name, indicated at SQLName 270, and the number of identical SQL insert statements, indicated at NoSQLs 272. In general, the larger the number of SQLs, the larger the workload contribution will be. SQL insert method 266 is typically generates outputs 268 in units of seconds of CPU usage and number of bytes transferred on the LAN. For one





number of columns returned, indicated at ColumnsReturned 292.

Select method 282 includes a selectivity criteria input 286, which can vary depending on the type of SQL select statement specified. In a preferred embodiment, the assumption is made that the select statements are indexed which is consistent with OLTP applications. In this same embodiment, for selects involving the joining of two or more tables, the selectivity criteria was geared toward most applicable cases for OLTP processing. Joins of more than three tables were not included since these cases do not comprise the majority of the transaction activity. Instead, the embodiment concentrated on providing most often occurring cases. To this end, the selectivity was defined for the cited embodiment as follows. If the select is a single table select, selectivity input 286 is the number of rows returned from the table. If the select is a two-table join, then selectivity criteria 286 is the number of rows selected from the outer, or left table, the inner table selectivity is assumed to be an average of four for each row selected from the outer table. If the select is a three-table join, then selectivity 286 is the number of values in the WHERE clauses that pertain to the outer table.

SQL select method 282 typically generates output 284 in units of seconds of CPU usage and number of bytes transferred on the LAN. The CPU usage is typically a complex function of the selectivity, including the table dimensions. For further discussion, see the experimental results section.

For each of the named SQL statements, a series of workload contributions 284, 280, 276, and 268 are generated and summed by method 252 and output as the total TxnComposition 264 which can have units of seconds of CPU usage. TxnComposition 264 can be multiplied by the transaction rate TxnRate 262 to arrive at the transaction workload contribution, TxnWorkloadContribution 258. Each

transaction can be named and specified in terms of the SQL statements, and the number of records and/or the selectivity criteria supplied. The total of all TxnWorkLoadContributions 258 can be summed together to arrive at a first estimate of total CPU utilization and total NIC utilization.

5       The value of the system configuration requirement can be obtained from the method SystemRequirementCalculations 294. The method 294 uses as input the two TxnWorkloadContribution components 258 and the input parameters MaxProcessorUtilization 296 and MaxNicUtilization 300. The method 294 produces the system configuration requirements 292. The system configuration requirements  
10   292 consists of the outputs #CPURequired 216, EffectiveCPUUtilization 218, MemoryRequirements 220, the LAN speed, indicated at CommsRequirement 304, the effective communications transfer rate, indicated at CommsRate 306, and the effective utilization of the NIC, indicated at EffectiveNICUtilization 308.

## EXPERIMENTAL RESULTS AND DETAILED METHODS

### 15       Measurement Definition and Methodology

A series of 14 SQL statements were defined for measurement on NT Servers running the database products SQL Server 6.5 and Oracle 8.04. These SQL statements, which were run against the tables defined for the 1 GB version of the TPC-D database, were thought to encompass several possible scenarios and business  
20   cases for the OLTP environment.

The series of SQL statements, consist of the following types:

- Inserts
- Deletes
- Updates

- Single table select: three selectivities
- Two table join: three selectivities
- Three table join: three selectivities

The intent behind this set of measurements was that any OLTP transaction can be thought of as some combination of the above SQLs or extensions of them. Thus, the sizer will solicit the user for the series of generic SQLs comprising each transaction. Based on curve fit estimates obtained from the measurement results, one can then estimate the CPU usage for each SQL and subsequently each transaction. By applying workload metrics to each transaction, one can then calculate the CPU utilization and subsequently, the CPU requirement.

The measurement of each SQL was run individually from a client, with no other load on the system. Each test consisted of one or more executions of the same SQL. A SQL may have been run several times in order to obtain a sufficient sample size in the collection of resource usage data. Further, each execution of a given SQL was written to cause the RDBMS to re-parse the SQL before executing it.

Scripts were developed to execute the series of SQLs and to cause certain RDBMS data to be collected and recorded. In order to implement re-parsing for each SQL execution, the script was written accordingly for each RDBMS. Specifically, SQL Server allowed looping and also parsed for each instance within the loop whereas Oracle appeared to parse only once within a loop; consequently, the Oracle SQLs could not be looped, and the scripts were written accordingly.

During each test, performance data was collected from the NT Performance Monitor on the server as well as from the RDBMS data collection construct (SQL Server's ISQL DBCC commands and Oracle's utlbstats/utlestats). This was after the

NT Server, the NT Performance Monitor, and the RDBMS were conditioned prior to each test.

### Measurement Results

Although several metrics were collected from each measurement, the metrics of most relevance to this version of the sizer were those that measured the CPU resource usage. This consisted of the following:

- Elapsed time
- Total per cent processor busy
- Per cent processor busy due to the RDBMS

Other statistics, such as logical reads and writes, and physical reads and writes, were also collected. However, this data was not used in the current version of the sizer; this was because all of the table accesses were indexed, and the number of logical IOs was always the number of B Tree levels plus data, consistent with an OLTP environment.

The processor data used in the sizer is the total processor usage, consisting of the RDBMS portion plus the kernel where the kernel activity is assumed to be attributed to the IO activity. In the current version of the sizer the CPU usage due to IO activity is not calculated separately.

Applicant believes that the total CPU time should be approximately the same for a given RDBMS, independent of the number of processors configured. The rationale is that the same amount of work is being done but that the work may be parsed out to a multiple number of processors; the exception is that there is some overhead to parse the work out so that possibly the total CPU time might increase slightly as a function of the number of processors configured. Results from most of

the Oracle measurements support this conjecture. It is expected that the same is true for SQL Server

### Analysis and Curve Fitting

#### Insert

5 For inserts, a single record was inserted into a table 10 times followed by a commit. This sequence was repeated 582 times. The CPU time (on a 200 MHz processor system) attributed to inserts, and used in the sizer, was then calculated as

$$\text{Seconds per record inserted} = (\text{Total CPU Time}) / 582 / 10$$

Values used in the sizer are:

	SQL Server	.004761
10	Oracle	.009

#### Delete

The deletes were measured during the process of reverting the database to its original state following the inserts. For the table used, an average of 3.88 records were deleted per delete SQL. In this case, 10 deletes were followed by a commit, and  
15 this process was repeated 150 times to return the table to its original condition. The CPU time attributed to deletes, and used in the sizer, was calculated as

$$\text{Seconds per delete SQL with 3.88 records deleted per SQL} = (\text{Total CPU time}) / 150 / 10$$

Values used in the sizer are:

	SQL Server	0.015416
20	Oracle	.015

#### Update

Three sets of measurements were conducted for update SQLs:

- A single SQL updated one record; this was followed by a commit
- A single SQL updated five records; this was followed by a commit

Each set of measurements was repeated 100 times.

SQL Server: For the sizer, the CPU time attributed to SQL Server updates was  
5 calculated based on the first two points, specifically:

$$\text{CPU Time} = 0.0012491 * (\text{No. records updated per SQL}) + .0004697 \text{ seconds}$$

Oracle: The results show fair amount of reduction in overall CPU time for  
updates as a function of the number of CPUs. The curve fitting for CPU time  
attributed to Oracle updates was calculated based on both the number of CPUs and the  
10 number of records updated per SQL, specifically:

$$\text{CPU Time} = -0.00248697 * (\text{No. CPUs}) + 0.00033591 * (\text{No. records updated per SQL}) + 0.00995405 \text{ seconds}$$

Currently the sizer pessimistically uses calculations based on 1 CPU.

#### Single Table Select - Indexed

15 Three single table SELECT SQLs were individually run. The WHERE  
clauses were set to produce selectivities of 2,588, 12,477, 24,896 records,  
respectively. These SQLs were defined so that the access method of the RDBMS was  
indexed, consistent with an OLTP environment. For SQL Server this was via a  
cluster key, thus, eliminating the additional IO to another leaf page if instead the  
20 index were non-clustered. For Oracle, this was via a standard index. The most  
suitable curve fit for both RDBMS seemed to be to divide the CPU time by the  
number of records selected, and then averaging that value. A value of .1 msec was  
given each RDBMS for parsing the SQL

The resulting formulas used, and applied in the sizer, were the following: .

SQL Server: (CPU Time) = .00006.73E \* (Records Selected) + .0001

Oracle: (CPU Time) = 0.00218 \* (Records Selected) + .0001

Note that the usual methods of curve fitting would apply to the selectivities noted, i.e., a very large number of records selected. Extrapolating to a few records  
5 selectivity (consistent with OLTP) produced either extremely large or extremely small CPU time overheads. As a result, the method of averaging time per record selected was chosen as a means of estimation.

#### Two Table Join - Indexed

For the two table join, an example of the business basis for the SQL SELECT  
10 was to find the various sources (e.g., prices) of a given item or list of items. In this case, there were on the average four such sources for each item. Three SQL SELECTs were constructed to return the sources for one, five, and 10 items, respectively. Results of the measurements for each RDBMS are shown in the charts below.

15 SQL Server: Curve fitting based on the template  $y=a*x^b$  produced the following results:

(CPU Time) = .0014665 \* (Items Selected) ^ .7552

Oracle: Curve fitting based on the template  $y=a*x+b$  produced the following results:

20 (CPU Time) = 0.00858 \* (Items Selected) + 0.003236

#### Three Table Join - Indexed

For the three table join, an example of the business basis for the SQL SELECT was to determine the status of orders placed on certain dates from a selected segment of the customer population. For this particular database, the customer segment chosen





The amount of data CommBytes transferred both ways across the LAN is estimated as follows per SQL:

Inserts:

CommBytes = 2 \* FrameHeader + SQLInsertIncomingData +  
5 AverageColumnsPerRow + SQLAcknowledge

Deletes:

CommBytes = 2 \* FrameHeader + SQLDeleteIncomingData +  
SQLAcknowledge

Updates:

10 CommBytes = 2 \* FrameHeader + SQLUpdateIncomingData \* NoRecords +  
SQLAcknowledge

Selects:

1 Table: Selected Bytes = Selectivity \* BytesPerColumn \* NoColumns  
2 Tables: Selected Bytes = 4 \* Selectivity \* BytesPerColumn \* NoColumns  
15 3 Tables: Selected Bytes = 5 \* Selectivity \* BytesPerColumn \* NoColumns  
CommBytes = FrameHeader + SQLSelectIncomingData  
Do While SelectedBytes > 1500 ' too big for a frame, break up into 1500  
bytes

CommBytes = CommBytes + TCPIPHdr + 1500  
20 SelectedBytes = SelectedBytes - 1500  
Loop  
If SelectedBytes > 0 Then  
CommBytes = CommBytes + TCPIPHdr + SelectedBytes  
End If  
25 nd

## HEADROOM CALCULATIONS

This SECTION describes a method for calculating the system configuration requirement and the headroom.

### 5    Processor Requirement

Define

TxnRate\_I    = transaction rate of transaction I

CPUTime\_I    = total CPUTime used to execute one instance of transaction I

TxnRate\_T    = sum of TxnRate\_I over all transactions

10            For each transaction, an input to method 294 is the transaction's effective contribution to CPU utilization, expressed as

$$(CPUTime\_I) * (TxnRate\_I)$$

Define

Util\_1CPU = sum of (CPUTime\_I) \* (TxnRate\_I) over all transactions

15            So Util\_1CPU is defined as the effective utilization of a single CPU. Note that this value can exceed 100%, implying that more than one CPU is required to service the transaction rate.

One method to determine the CPU requirement NoCPUs is as follows:

$$NoCPUs = CEILING(Util\_1CPU/100)$$

20            While (Util\_1CPU/100)/NoCPUs > MaxProcessorUtilization

$$NoCPUs = NoCPUs + 1$$

Wend

We note, however, that adding multiple processors does not increase performance with 100% efficiency. Thus, for example, by doubling the number of  
25    processors, we achieve anywhere from 150% to 180% as much performance, rather

than 200%. Much of this is due to a combination of the instruction speed slows due to locality of reference and the number of instructions per transaction increase due to increased housekeeping overhead.

The service time per transaction is the quotient (instructions per transaction) / (instructions per second). It has been found in studies that the service time per transaction is a linear function of the number of processors, i.e., service time per transaction,  $n$  processors = (service time, 1 processor) \* ( $a * n + b$ ) where  $a$  and  $b$  are constants determined from analysis of measurement results. Thus, the transaction rate is proportional to the reciprocal, that is,

10 transaction rate  $\sim 1 / (a * n + b)$

For this version of the sizer, the values  $a$ ,  $b$  are:

$$a = 0.028982$$

$$b = 0.973703$$

The above algorithm is then modified as follows:

15 Define

$$\text{NoCPUs} = \text{CEILING}(\text{Util\_1CPU}/100)$$

$$\text{CPUSvc\_1x} = \text{Util\_1CPU} / \text{Txn\_Rate\_T}$$

bResult = False

While Not bResult

20 temp = NoCPUs / ( $a * \text{NoCPUs} + b$ ) \* MaxCPUUtilization

If temp < Util\_1CPU Then

$$\text{NoCPUs} = \text{NoCPUs} + 1$$

Else

bResult = True

25 EffectiveCPUUtilization = Ceiling( $((a * \text{NoCPUs} + b) * \text{Util\_1CPU} / \text{NoCPU} * 100)$ )

[illegible]

5

The EffectiveCPUUtilization in the above calculations does not exceed the MaxCPUUtilization, thus allowing the user to select a MaxCPUUtilization which is lower than required for good performance, thus allowing “headroom” for growth.

10

### LAN Speed Requirement

The LAN speed requirement is determined in a manner similar to that described above.

Define

15

**NICType** = bandwidth of NIC card selected

CommReq = bytes per second required for all of the transactions

**MaxNICUtil(i)** = Recommended maximum utilization of NIC card i

20

For  $i = 1$  To Number of Card Types, starting with lowest capacity

$$\text{Temp} = \text{NIC\_Speed}(i) * \text{MaxNICUtil}(i) / 8 / 1000000$$

If CommReq  $\leq$  Temp Then

```
NICType = NIC_Speed(i)
```

25

End If

Next

If NICType = 0

5                   Message:       "The communications requirement exceeds the effective  
  maximum imposed by the utilization limit for all  
  LAN speeds."

End If

                  This algorithm also demonstrates the headroom algorithm.

#### Memory Calculations

10                Because the database servers are of the enterprise quality, the amount of  
memory required is generally very high. As a rule of thumb the memory requirement  
has been specified as an additional 512 MB per processor up to the limit imposed by  
the machine and the operating system.

                  Numerous advantages of the invention covered by this document have been  
15   set forth in the foregoing description. It will be understood, however, that this  
disclosure is, in many respects, only illustrative. Changes may be made in details,  
particularly in matters of shape, size, and arrangement of parts without exceeding the  
scope of the invention. The invention's scope is, of course, defined in the language in  
which the appended claims are expressed.